

Applying a Component Model to Grid Application Services

Rainer Schmidt, Siegfried Benkner, Ivona Brandic, Gerhard Engelbrecht
Institute of Scientific Computing, University of Vienna
Nordbergstrasse 15/C/3, A-1090 Vienna, Austria
email: rainer@par.univie.ac.at

Abstract

Programming client-side applications based on composing units of distributed software is an important research issue in the Grid computing community. In this context, it is a key requirement to provide application developers with high-level programming models and frameworks which are well supported by the underlying Grid architecture. In this paper we present an approach that applies a component model to the Vienna Grid Environment, a Web service based Grid environment, which enables the provision of parallel applications as QoS-aware Grid services, whose performance characteristics may be dynamically negotiated between a client application and service providers. Our environment comprises Grid services, a distributed component framework, and a component-based, client-side programming framework. Our component model follows the Common Component Architecture and allows to explicitly express and dynamically manage context dependencies with respect to the hosting environment, computational resources, as well as dependencies on other components. This mechanism serves as a basis for independent service deployment, service selection and composition. Our work can be seen as a first step towards a component-based programming model for service-oriented Grid infrastructures utilizing standard Web services technologies.

Index Terms

Web Services, Component Software, CCA, Component-oriented-programming, Grid Computing, Service-oriented architecture

I. Introduction

In recent years, Web service technology has gained more and more importance in the area of Grid Computing. The Open Grid Service Architecture [1] developed within the Global Grid Forum has motivated Grid architects to build environments based on a service-oriented architecture utilizing Web service technology. A significant development in this direction was the evolution of the Globus toolkit [2] towards the Web Service Resource Framework [3]. While efforts are on the way aiming to develop a general architecture for computational Grids (OGSA [4], OMII [5]), there is currently little consensus on suitable programming paradigms for Grid application development [11].

Grid middleware is still very complex to use and often requires users having a detailed insight into the Grid infrastructure. This drawback motivates the development of appropriate programming models for Grid environments. Grid computing requires methodologies allowing end users to construct composite Grid applications based on basic operations such as creating and interconnecting remote resources. However, Grids are mostly built following a service-oriented architecture using Web services technology which has not been designed to fit the idea of a component-based plug-and-play client programming framework. Services are typically discovered dynamically rather than created, they further do not provide means to describe dependencies for example on other services. Web service technology provides a versatile messaging facility but currently lacks an extensive component model applicable to service composition. From a client's point of view we argue (as discussed in [6]) that a component-based programming would be a well suited paradigm for client-side

Grid applications construction. In this paper we present an architecture and prototype implementation that applies such a component model to a Web service based Grid infrastructure. The system represents distributed application services as composable components attempting to satisfy the component definition given in [7].

Our component model follows the Common Component Architecture (CCA) [8] and allows to explicitly express and dynamically manage context dependencies [7] concerning the hosting environment, computational resources, as well as dependencies on other components. We therefore provide defined interfaces and mechanisms to configure and deploy a VGE component.

Our infrastructure is based on the Vienna Grid Environment (VGE) [9], a Grid infrastructure for the provision of parallel applications as Grid services over standard Web service technology. The VGE service provision framework is currently being utilized in the GEMSS Project [10] for the Grid provision of advanced medical simulation services. Most of these applications consist of various steps of execution (e.g. mesh generation, data analysis, visualization) which can be separately deployed as Web services using VGE.

We describe an approach allowing a well defined configuration and independent deployment of VGE application Web services. We further explain how service dependencies (e.g. on components, data) can be gathered by a CCA based distributed component framework and being utilized for Grid application composition. By integrating a component model, we want to provide clients with a component-based programming environment and hide implementation details of the service-oriented Grid system.

The remainder of the paper is organized as follows: Section II briefly explains the VGE service provision framework. The VGE component model is examined in section III. Section IV describes the distributed component framework and Section V sketches the client-side programming model. A discussion on related work is given in section VI. Conclusions and future work are presented in section VII.

II. VGE Service Provision

As a key feature, VGE supports a flexible QoS negotiation model which enables clients to negotiate dynamically various QoS guarantees with potential service providers. In order to provide support for performance guarantees, a VGE service relies on an application-specific performance model to estimate the execution time for a specific service request. In the following we briefly describe the architecture of VGE services and the service provision environment. For a more detailed description of VGE the reader is referred to [9].

A. Generic Application Web Services

At its core, a VGE application service follows a generic application service model and exposes a native application as a Web service to be accessed by multiple remote clients over the Internet.

The application service provides generic operations for job management, data staging, and optional operations for error recovery and QoS support. Figure 1 shows the interfaces and operations a VGE service provides.

The operations upload and download are used for uploading of input data to a service and downloading of output data. The operation push is provided for data staging between different services. The execution of a remote application can be initiated by calling start and stopped by calling kill. The operation getStatus allows the client to download an application-specific status file. The operations provided in the error recovery interface enable clients to control checkpointing and restarting of applications. The operations of the interface QoS are optionally used during QoS negotiation. The behavior of these operations is customized for a specific application by means of an XML application descriptor.

B. Application Descriptor

An application descriptor is an XML document that provides meta-data about an existing application and its computational resource. The information provided in an application descriptor defines the semantics of the operations of a generic application service and enables the VGE service provision framework to expose an application automatically as a Web service. The application descriptor is the application specific part of the service environment, interpreted by the service provision framework and mapped against the generic methods. VGE further provides descriptors concerning Web service technology which allows configuring concerns related to Web service deployment (e.g. provided interfaces) or WS-security.

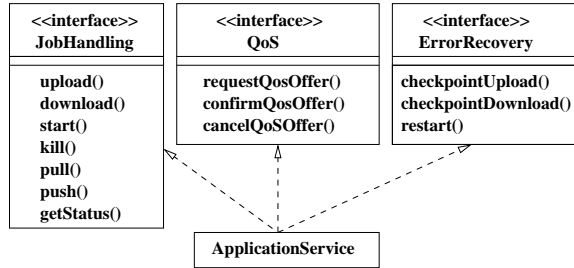


Fig. 1. Generic Application Service

An application descriptor usually specifies the input/output files, the script for initiating job execution, and scripts for gathering status information. The `compute-resource-manager` element may be used to specify an interface to a resource scheduler. Currently NEC's COSY [13] and the MAUI [14] scheduler are being utilized. For QoS support, e.g. execution time guarantees, a set of request parameters, for specifying request meta data, and a set of machine parameters have to be specified. In order to allow dynamic service selection based on performance characteristics, a client may invoke the operation `requestQoSOffer` and supply actual values for the request parameters. These request parameters are fed into the machine-specific performance model to obtain an estimate of the execution time. In order to support service selection based on performance characteristics, the framework may initiate a QoS negotiation with multiple service providers by invoking corresponding operations of the QoS interface (see Figure 1).

Figure 2 shows a simplified excerpt of an application descriptor. Information such as input/output files, data formats, etc. are required for interconnecting services and have to be communicated to the client framework by the component system.

C. Web Service Technology

VGE makes use of open-source frameworks such as Tomcat [15] and Axis [16] to provide HPC applications over standard Web Service technology (SOAP, WSDL, WS-Security). For large file transfers SOAP attachments are utilized. QoS contracts are formulated as an XML document following the Web Service Level Agreement [17] (WSLA) specification. An operational PKI infrastructure based on X.509 certificates provides transport and message layer security. The application services are implemented as stateless Web services acting upon a stateful resource (e.g. an application installed on a computer cluster), which is conceptually equivalent to the mechanisms defined by the Web Service Resource Framework [12].

III. A Component Model for Grid Services

In general, Web services are highly self-contained as they only expose information relevant to accessing their ports but do not provide ways to specify dependencies to the hosting environment or to other components. This characteristic has the advantage that standalone applications can be easily made available as services. If a Web service encapsulates logic which has local dependencies (e.g. to the operating system) that cannot be expressed in a uniform way, the service may not be deployable without cumbersome adaptations for different hosting environments. The concept of `uses ports` like it is used by the Corba Component Model (CCM) [18], or the CCA, which allows a component to specify dependencies to `provides ports` of other components, is currently not addressed by Web service technology. Integrating such a mechanism would allow a service to make implicitly use of a remote computation at runtime, for example retrieving data from a remote monitoring service, which might be found dynamically or configured at deployment time. Furthermore, component-based-programming by interconnecting `provides` and `uses ports` could be explicitly utilized via a client programming environment, which provides a simple and efficient way to construct composite applications.

For VGE Web services we are developing a component model (Figure 3) that provides interfaces for configuring parameters concerning the Web service (e.g. security level) as well as providing meta-data describing the wrapped native application installed on the computational resource (see Section II-B). The VGE component model further supports the concept of `provides` and `uses ports` and fits into a CCA based distributed component framework that manages

```

<application>
  <configuration>
    <working-directory>
      <path>/home/...</path>
    </working-directory>
    <input-files>
      <file>
        <name>TiH2.in0</name>
      </file>
      ...
    </input-files>
    <output-files>
      <file>
        <name>TiH2.out0</name>
      </file>
      ...
    </output-files>
    <job-script>
      <path>/home/rainer/...</path>
    </job-script>
    <finish-flag>
      <name>finish</name>
    </finish-flag>
  </configuration>
  <performance>
    <performance-parameters/>
    <machine-parameters>
      <number-of-nodes>...
    </machine-parameters>
    ...
    <provider-parameters>
      <compute-resource-manager-class>at.ac.univie.iss...</compute-resource-manager-class>
      <performance-model-class>at.ac.univie.iss.apm...</performance-model-class>
    </provider-parameters>
  </performance>
</application>

```

Fig. 2. Application Descriptor

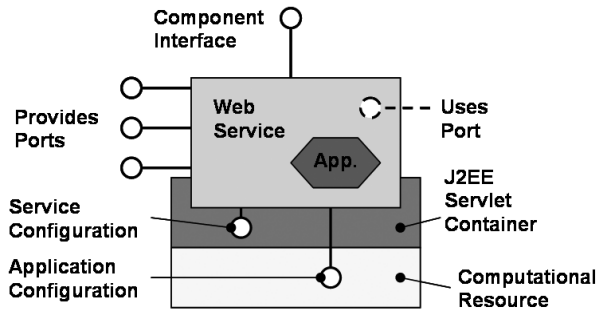


Fig. 3. VGE Component Interfaces

the components and allows to dynamically register and remove their ports. For deployment VGE services are packaged into a Web Application Archive which can be easily deployed into a J2EE Servlet Container without requiring additional software being installed. Our model provides interfaces for component configuration and dependency specification, which are described below.

Application Configuration Interface

Native applications are deployed as VGE services based on an XML application descriptor(II-B). The descriptor contains data required for job and data handling (e.g. working directory, i/o data, library configuration) as well as optionally information related to Quality of Service (e.g. compute resource manager, performance model). A deployment tool supports the user to create the application descriptor and may automatically package, deploy, and register the Web service.

Service Configuration Interface

This interface allows configuring properties related to Web service technology based on an XML document. It provides for adjusting the level of Web service security (signature, encryption, authentication and authorization) a service exhibits. Security is based on X.509 certificates and implemented by message handlers. Furthermore, it is required to specify the ports a service provides as well as dependencies to other services in form of `uses ports`. The provision of an additional component interface enables the service being plugged into the component framework.

Component Interface

The component interface provides the functionality required to configure a service in order to communicate with a particular component framework and to make use of the framework services. It allows a service to register itself as component with associated meta-data and actively register and retrieve ports. The component functionality is provided as library and can be added to a Web service easily by extending the deployment descriptor.

Provides and Uses Ports

VGE services may expose interfaces for job handling, Quality of Service negotiation, and error recovery. For supporting large input/output data, VGE Grid services use file transfer via SOAP attachments for data exchange instead of communicating via XML encoded documents. Based on the component model it is also possible to incorporate non VGE services with different ports and dependencies to other services into the environment. A service may register arbitrary `provides` and `uses ports` and dynamically retrieve them from the framework, in a similar way a client application uses remote ports.

IV. Integrating Web Services with a Component Framework

The VGE component model is based on the Common Component Architecture specification. It adopts concepts like `provides/uses ports`, component interface, framework services and builder service. A CCA framework implements defined interfaces that rule component interactions and provides the environment for components to run and communicate. Our component framework used for VGE services is implemented as Web service itself and allows the integration of Web services, which follow our component model, into the component environment (Figure 4). The idea is it to provide a component-oriented programming environment allowing clients to compose application from services, while the complexity of the service-oriented Grid infrastructure is hidden. The framework acts as runtime broker between the application demands and available Grid services, and therefore has to select the appropriate resources and establish the connections among them.

A. Overview of System Dynamics

The component interface provides mechanisms that allow a Grid service to contact the component framework. Services provided by the framework can be used by components to register their provided/required ports as well as to request them

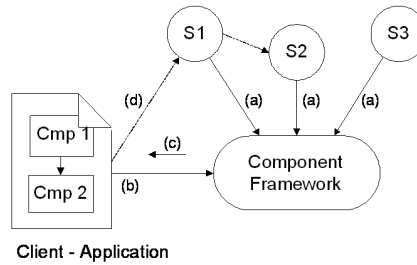


Fig. 4. Overall System Architecture: The component-based client application is dynamically executed on available Grid services.

dynamically at execution time (a). The client application may comprise several components, which can be instantiated, connected and invoked. The Builder service provided by the framework locates the services on behalf of the client application at execution time (b) and delivers runtime relevant service information (e.g. classloader, i/o characteristics) back to the involved entities (c) which allows them to interact (d).

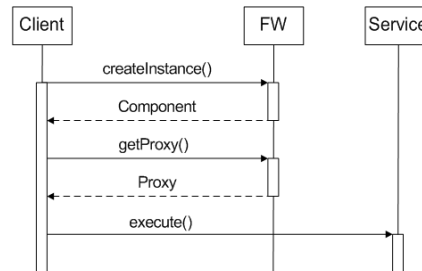


Fig. 5. Dynamic Service Invocation: The creation of a component instance may require a client to download "glue code" to the service from the framework.

B. Mobile Proxy Mechanism

When registering a component with the framework it is possible to associate a proxy implementation with it, which is stored within a proxy repository at the framework. The repository is used to provide the client environment with a high flexibility concerning dynamic service selection and invocation using a mobile proxy mechanism. This is an important issue, especially for Grid Computing, when data structures sent via SOAP are not XML encoded (e.g. for the reason of performance), disabling clients to automatically generate stub code from the WSDL interface. Implementing mobile proxies enables our client applications to stay compliant with different versions of VGE services by dynamically retrieving proxy updates, and allow to incorporate non-VGE services if they follow the component model. We therefore extended the Java classloader with a dynamic proxy pattern that allows the client to access remote component libraries from the framework (Figure 5). At execution time the client environment contacts the framework to fill the component representation with substantial runtime information (component id, service endpoint, proxy class name, class loader). Invoking a method on a component port causes the runtime engine to generate a call to a service using the local code base. In case of a fault or if the component is unknown to the client-side code repository, the Java classloader dynamically retrieves an appropriate proxy file from the framework where the component is registered. Dynamic binding mechanisms using interface based service discovery and proxy lookup which we have applied for Web services are well known from Jini [19] technology.

V. Component based Programming

The client programming environment is currently provided as Java API and based on the CCA builder service interface. It offers a component-based programming model that allows the construction of composite Grid applications based on

composable entities. The available resources are virtualized behind the component abstractions and are selected by the component framework during application execution. This section describes elementary concepts and operations provided by the client environment.

Describing the Component: As resources offered via Grid Services (especially if they have generic interfaces) cannot be specified by class or interface names uniquely, it is required to describe them also by their characteristics. For our environment, VGE components can be described by associated meta-data such as name and version of the underlying application, input/output data format, etc. To also allow the integration of non VGE components into the environment (e.g. a database adaptor) it is further required to specify the interfaces a component has to provide by an interface descriptor. Such a programming model provides the advantage that the described resources can be virtualized by the framework and selected at runtime on behalf of the client.

Component Creation: The `createInstance()` method instantiates a component object based on a component description. The client therefore retrieves a handle to a service and optionally proxy code from the component framework. Invoking this method further causes the creation of a client session and endpoint reference on the computational resource (see WS-Resource Creation [12]). Components are being identified by a `ComponentId` which consists of the service handle and a session identifier. Thus, the creation of a Web service component results in the creation of a session for the client application on the resource which can be accessed via a component identifier.

Component Connection: Web Service Components may have internal dependencies on other components or can be connected explicitly by the client application using a `connect()` operation. The connections between the components could be seen as the pipes within a Pipe-And-Filter architecture. After an application has processed the input data and indicates that the calculation has finished, the output files are piped by calling a `push()` operation to a target service for further processing. The generated output files are specified within the application descriptor at the service and can be identified at the client side by corresponding attributes stored in the component object. By assigning output files to the connections using these properties it is possible to attach multiple connections to a component.

Invoking Provides Ports: Components provide access to their ports via the generic operation `GetComponentPort()`. The component therefore generates a proxy to a `provides port` of a component which is represented as object of the requested interface data type. This port representation allows to invoke an interfaces provided by a remote service and may automatically contact the component framework for proxy code if it is unknown to the client.

Decomposing the Application: The operation `destroyInstance()` destroys the component within the client application and invalidates the session at the target resource. Destroying the client session at the computing resource will automatically delete all permanently stored files and optionally resource reservations.

VI. Discussion of Related Work

Distributed component models such as Corba [18], EJB [20] or DCOM [21] are widely used mostly in the context of commercial applications. The work on GridCCM [22] extends to the Corba Component Model supporting parallel component communications. The Common Component Architecture developed within the CCA forum [23] defines a component model for high-performance computing based on interface definitions. Ccaffeine [24] provides a framework (based on CCA) for component-based programming in the area of scientific and cluster computing that supports high-performance MPI components and uses Babel for language interoperability between third party libraries. XCAT3 [25] is a distributed framework that accesses Grid services e.g. OGSF [1] based on CCA mechanisms. It uses XSOAP [26] for communication and can use ssh or Globus GRAM [2] for remote component instantiation. Several projects, mostly in the workflow community, are concerned with providing/virtualizing access to Web services in order to compose applications from distributed resources (e.g. Triana [27], Taverna [29], Kepler [28]). Workflow systems, however, can be seen on top of distributed architectures and programming models as they somehow provide concepts that allow end users to utilize these environments.

VII. Conclusion

We presented an environment that adds the abstraction of components to a Web service based Grid infrastructure. Our component model is based on the Common Component Architecture and applied to VGE application services allowing an explicit specification of context dependencies. We further provide a distributed component framework as well as a component-based client programming environment. We expect this work not only relevant to the Grid community but also to

people from the Web service and component-based software engineering communities as we did not purely address concerns related to Grid computing but also discussed architectural issues concerning Web service configuration, deployment, and interoperability as well as a component-oriented programming model. Our implementation of the component model is still in prototype stage which requires further experiments and analysis being done using real world applications in order to evaluate the potential of this concept. For future work we plan to provide more sophisticated support for service selection and composition based on QoS properties, in particular execution time guaranties.

References

- [1] Foster, I., Kesselman, C., Nick, J., Tuecke, S. "*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*", Globus Project, 2002. Available at <http://www.globus.org/research/papers/ogsa.pdf>.
- [2] The Globus Alliance. <http://www.globus.org>.
- [3] The Web Service Resource Framework. <http://www.globus.org/wsrf/>.
- [4] The Open Grid Services Architecture, Version 1.0. <http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf>.
- [5] The UK e-Science Open Middleware Infrastructure Institute. <http://download.omii.ac.uk/omii/>
- [6] Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthakrishnan, Felipe Bertrand, Ken Chiu, Matt Farrellee, Madhu Govindaraju, Shriram Krishnan, Lavanya Ramakrishnan, Yogesh Simmhan, Alek Slominski, Yu Ma, Caroline Olariu, Nicolas Rey-Cenevaz. "*Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications*" Journal of Cluster Computing, 2002.
- [7] Clemens Szyperski with Dominik Gruntz and Stephan Murer "*Component Software: Beyond Object-Oriented Programming. Second Edition*" Addison-Wesley and ACM Press, 2002.
- [8] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. "*Toward a Common Component Architecture for High-Performance Scientific Computing*" Proceedings of the High-Performance Distributed Computing Conference, August 1999, pp. 115-124.
- [9] S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. "*VGE - A Service-Oriented Environment for On-Demand Supercomputing*", Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.
- [10] Gemss Project homepage. <http://www.gemss.de>.
- [11] M. Parashar, J.C. Browne. Conceptual and implementation models for the grid. Proceedings of the IEEE, March 2005 Vol 93, Issue 3. pages 653 - 668.
- [12] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. F. Ferguson, F. Leymann, I. Sedukhin, M. Nally, D. Snelling, T. Storey, W. Vambenepe, S. Weerawarana. "*Modeling Stateful Resources with Web Services*", Version 1.1, <http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf>, May 2004.
- [13] The NEC COSY Job Scheduling System. <http://www.ccr1-nece.de/falk/COSY/cosy.shtml>.
- [14] Maui Cluster Scheduler. <http://www.clusterresources.com/products/maui/>
- [15] Apache Tomcat. <http://jakarta.apache.org/tomcat/>.
- [16] Apache Axis. <http://ws.apache.org/axis/>.
- [17] Web Service Level Agreement (WSLA) Language Specification. <http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf>, IBM 2001-2003
- [18] CORBA Component Model, v3.0, OMG. <http://www.omg.org/technology/documents/formal/components.htm>.
- [19] J. Waldo. "*The Jini Architecture For Network-Centric Computing*" Communications of the ACM, 42(7):76-82, July 1999.
- [20] Enterprise JavaBeans technology: <http://java.sun.com/products/ejb>.
- [21] COM Component Object Model Technologies, Microsoft, <http://www.microsoft.com/com/default.msp>.
- [22] C. Prez, T. Priol, A. Ribes. "*A Parallel CORBA Component Model for Numerical Code Coupling*" The International Journal of High Performance Computing Applications (IJHPCA), 17(4):417-429, 2003.
- [23] The CCA Forum. <http://cca-forum.org/>.
- [24] B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt, J. A. Kohl. "*The CCA core specification in a distributed memory SPMD framework*" Concurrency and Computation: Practice and Experience 14(5), 2002.
- [25] S. Krishnan, D. Gannon. "*XCAT3: A Framework for CCA Components as OGSA Services*" Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, April 2004, pp. 90-97.
- [26] XSOAP toolkit. <http://www.extreme.indiana.edu/xgws/xsoap>.
- [27] <http://www.trianacode.org/>
- [28] <http://kepler-project.org/>
- [29] <http://taverna.sourceforge.net/>
- [30] H. Liu, M. Parashar and S. Hariri. "*A Component-based Programming Framework for Autonomic Applications*" Proceedings of the 1st IEEE International Conference on Autonomic Computing (ICAC-04), IEEE Computer Society Press, New York, NY, USA, May 2004. International Conference on Mathematics and Engineering Techniques in Medicine and Biological Science, p. 216-221, Las Vegas, 2003.